

# メディアアート・プログラミング2

東京藝術大学 芸術情報センター開設科目 後期金曜4限 第4週

2023.10.27 松浦知也 ([matsura.tomoya@noc.geidai.ac.jp](mailto:matsura.tomoya@noc.geidai.ac.jp) [teach@matsuuratomoya.com](mailto:teach@matsuuratomoya.com))



# シェルコマンドとパイプ

# シェルコマンドとパイプ

- 1つのコマンドは普通標準入力(stdin)をターミナルから受け取り、結果を標準出力(stdout)に返す
- “コマンドA | コマンドB” のようにすると、コマンドAの出力をそのままコマンドBの入力に渡すことができる

```
say "こんにちは"
```

```
echo "こんにちは" | say
```

※sayコマンドはmacOS特有のコマンド

# ところで...

- lsコマンドで表示できるのは必ずしもFinderから見えるファイルだけではない
  - というより、ファイルとデバイスを等価に扱えるようにしているのがOSの大きな仕事の1つ
  - 例えば、システムデバイスとかは `ls /dev` とかすると覗ける
  - `head /dev/urandom` とかするとシステムの乱数生成機が使える
- 昔のLinuxには `/dev/dsp`というパイプでバイナリデータを投げるとシステムのオーディオドライバに直接波形を流し込める擬似デバイスが存在した
  - 今もLinuxでは`aplay`というコマンドで同等のことができる
  - Macでは`afplay`という類似コマンドはあるものの、パイプ入力は受け付けてない

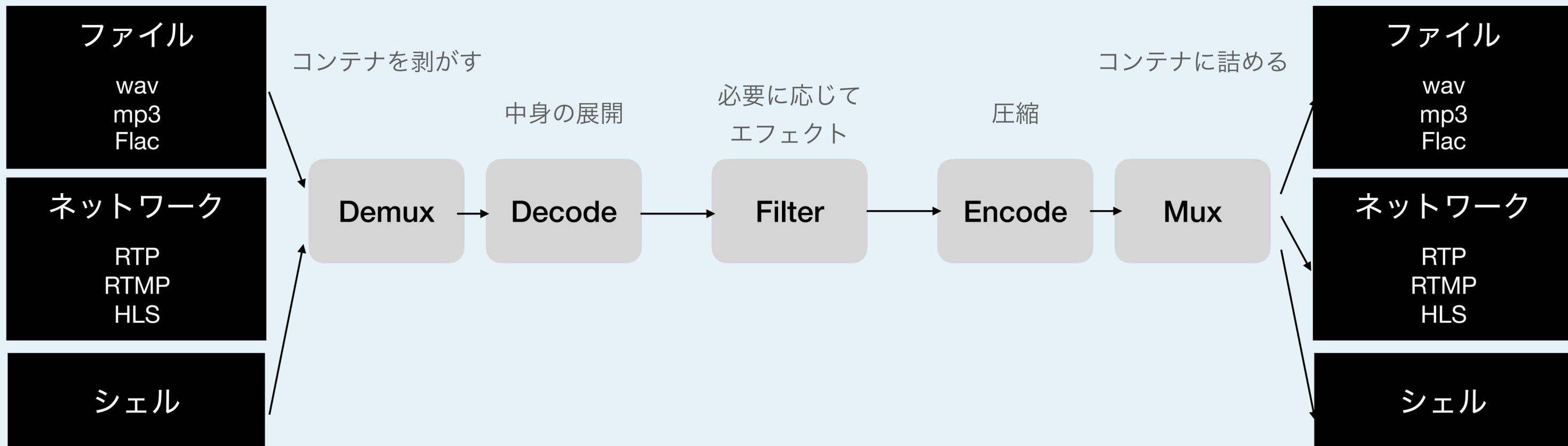
# 今日使うツールのインストール

```
brew install ffmpeg
```

- macOSでは/dev/dspやaplayの代わりにffmpegを使用すると等価なことができる

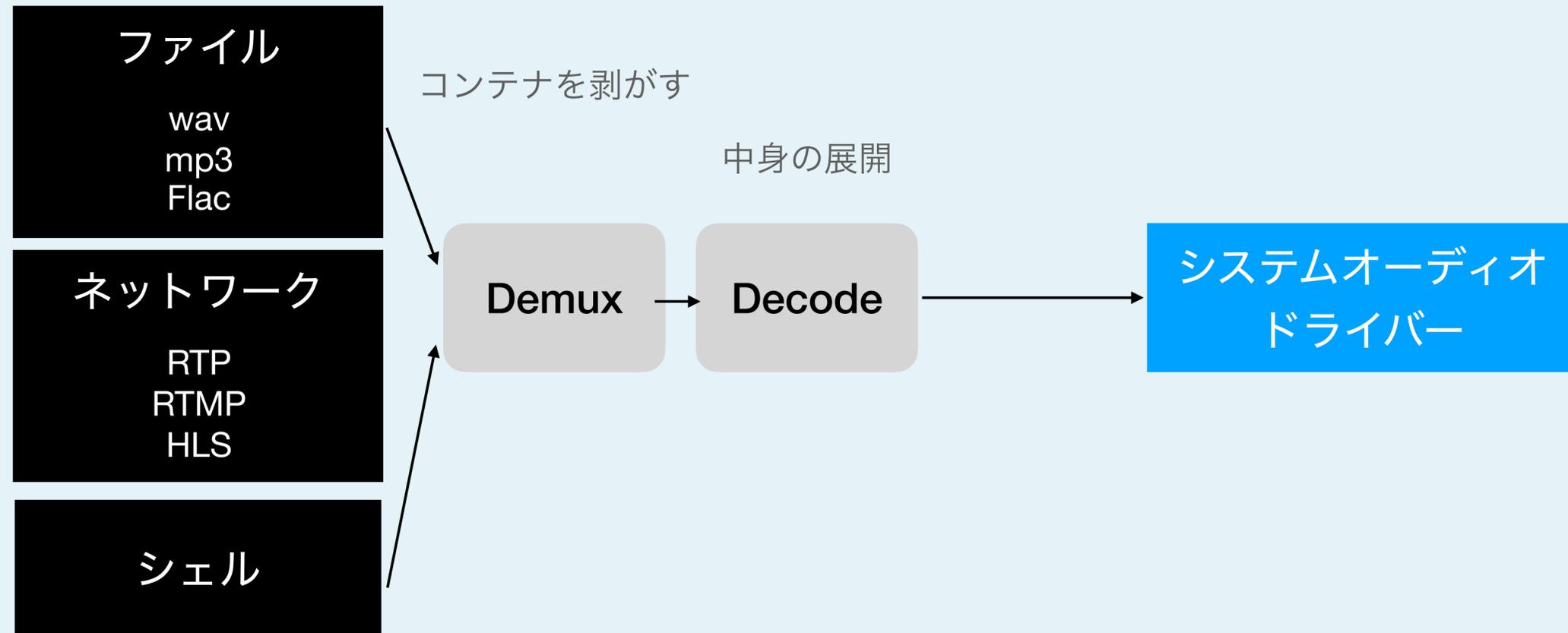
# ffmpegとは

- 一般的には、映像・音声ファイルフォーマット変換ツール
- 正確には、いろんな種類のビデオ/オーディオストリームを変換するツール
- いろんな配信サービスの裏では大抵こいつが動いている
- コマンドのオプションの種類が大変多いので、わからなくなったらChatGPTなどに聞く



# ffplay

- ffmpegの機能を使って、ファイルをシステムオーディオで再生できるアプリ
- ffmpegがインストールされれば自動で入ってくる



# ffplayでパイプを聴く

- 前回やった「AudacityでAudacityを聴く」をffmpegからやってみよう

```
cat '/Applications/Audacity.app/Contents/MacOS/Audacity' |  
    ffplay -f u8 -i pipe:0 -ar 44k -ac 1
```

catはあらゆるファイルの中身を出力するコマンド

- スペクトログラムの再生ウィンドウが自動で立ち上がるが、wキーで波形表示と切り替えが可能。  
再生が終わってもウィンドウが閉じないので、ctrl+Cでコマンドを止める

```
ffplay -f u8 -i pipe:0 -ar 44k -ac 1
```

フォーマットは例えば

u8: Unsigned 8-bit PCM

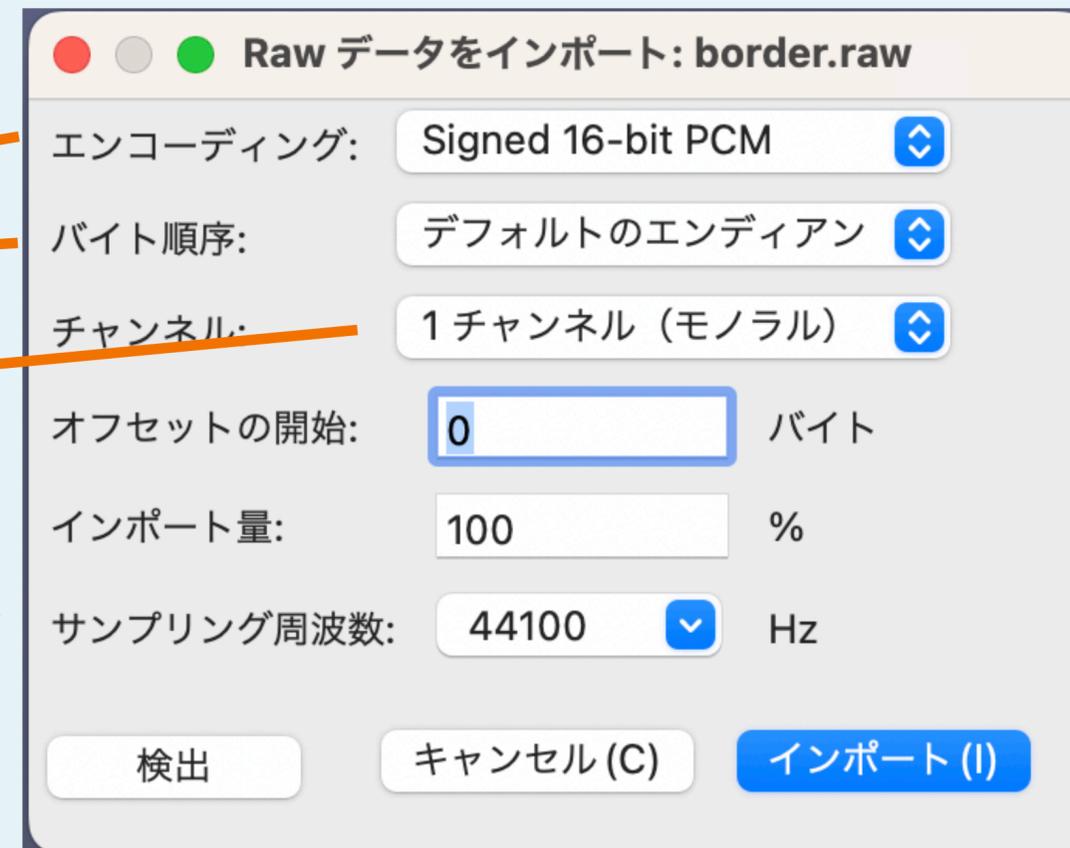
s16le: signed 8-bit PCM Little Endian

s16be: signed 8-bit PCM Big Endian

s24le: signed 24-bit PCM Little Endian

※エンディアンとはビット列の数えかたをどちら向きに数えるかの概念。

WindowsとMacでデフォルトが違う



Bytebeat-

データを直接生成して聴く

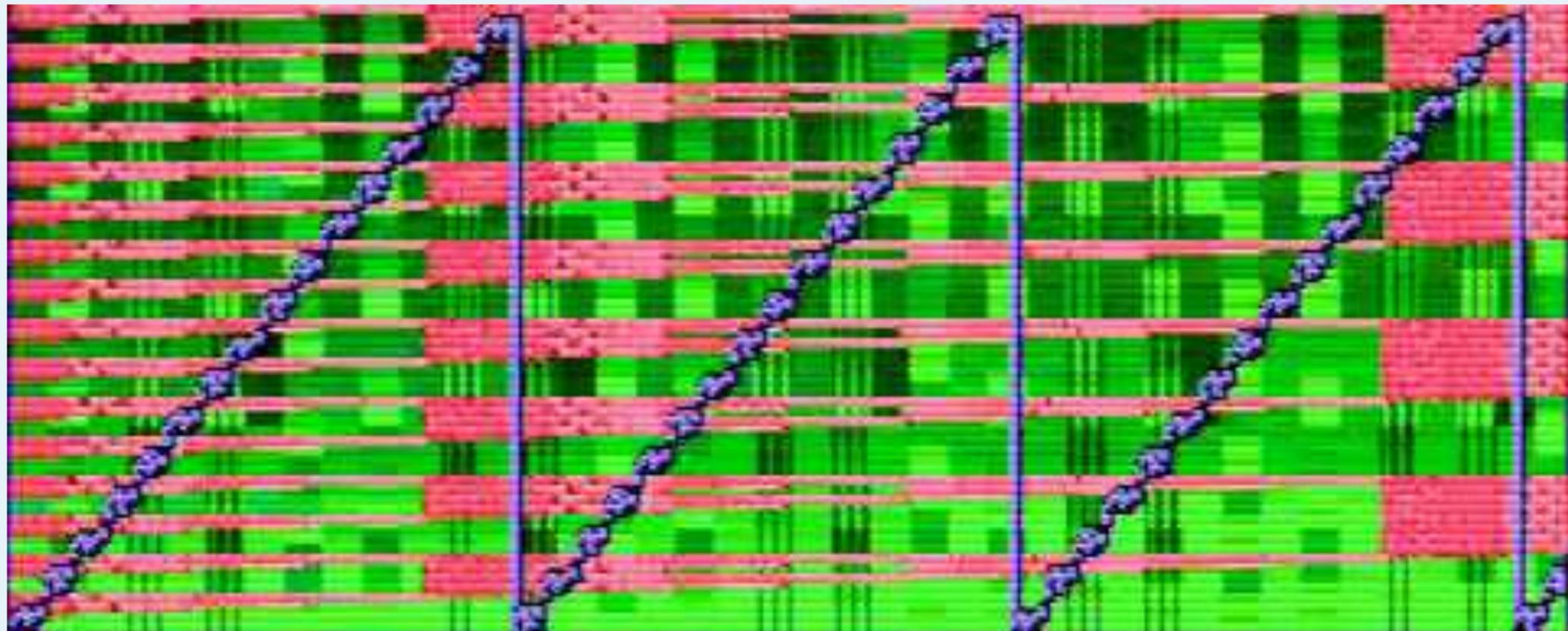
# Bytebeatとは

- 1行のC言語プログラムを使ったコンピューター音楽。
  - 2011年にviznutが動画とブログで発表してから様々なユーザーが投稿
- 1バイトのデータを標準出力に書き込むputchar関数を使うシンプルなプログラムを、パイプで/dev/dspやaplayに流し込むことで音を鳴らす
- プログラム (=曲) は例えばこんな感じ

```
(t >> 10 ^ t >> 11) % 5 * ((t >> 14 & 3 ^ t >> 15 & 1) + 1) * t  
% 99 + ((3 + (t >> 14 & 3) - (t >> 16 & 1)) / 3 * t % 99 & 64)
```

ここでtは0から1サンプルずつ増えていく整数 (多分32bit)

Fractalized Past by: lhphr  
HTML5 Bytebeatのexamplesより



```
main(t){for(;;t++)putchar(  
((t%("36364689"[t>>13&7]&15))/12&128)  
+((((((t>>12)^(t>>12)-2)%11*t)/4  
|t>>13)&127)  
);}
```

From: ryg  
44.1 kHz output rate

# countercomplex

bitwise creations in a pre-apocalyptic world

SUNDAY, 2 OCTOBER 2011

## ➔ Algorithmic symphonies from one line of code -- how and why?

Lately, there has been a lot of experimentation with very short programs that synthesize something that sounds like music. I now want to share some information and thoughts about these experiments.

First, some background. On 2011-09-26, I released the following video on Youtube, presenting seven programs and their musical output:



This video gathered a lot of interest, inspiring many programmers to experiment on their own and share their findings. This was further boosted by [Bemmu's on-line](#)

Countecomplex is Viznut's blog.



[View my complete profile](#)

Demoscene, hardcore algorithmic arts, pan-hackerdom, radical low-complexity, digital degrowth.

#countercomplex @ IRCnet

[viznut.fi](http://viznut.fi)

Blog Archive

▶ 2018 (1)

▶ 2015 (3)

▶ 2014 (3)

▶ 2013 (2)

▶ 2012 (2)

▼ 2011 (12)

▶ December (1)

▶ November (1)

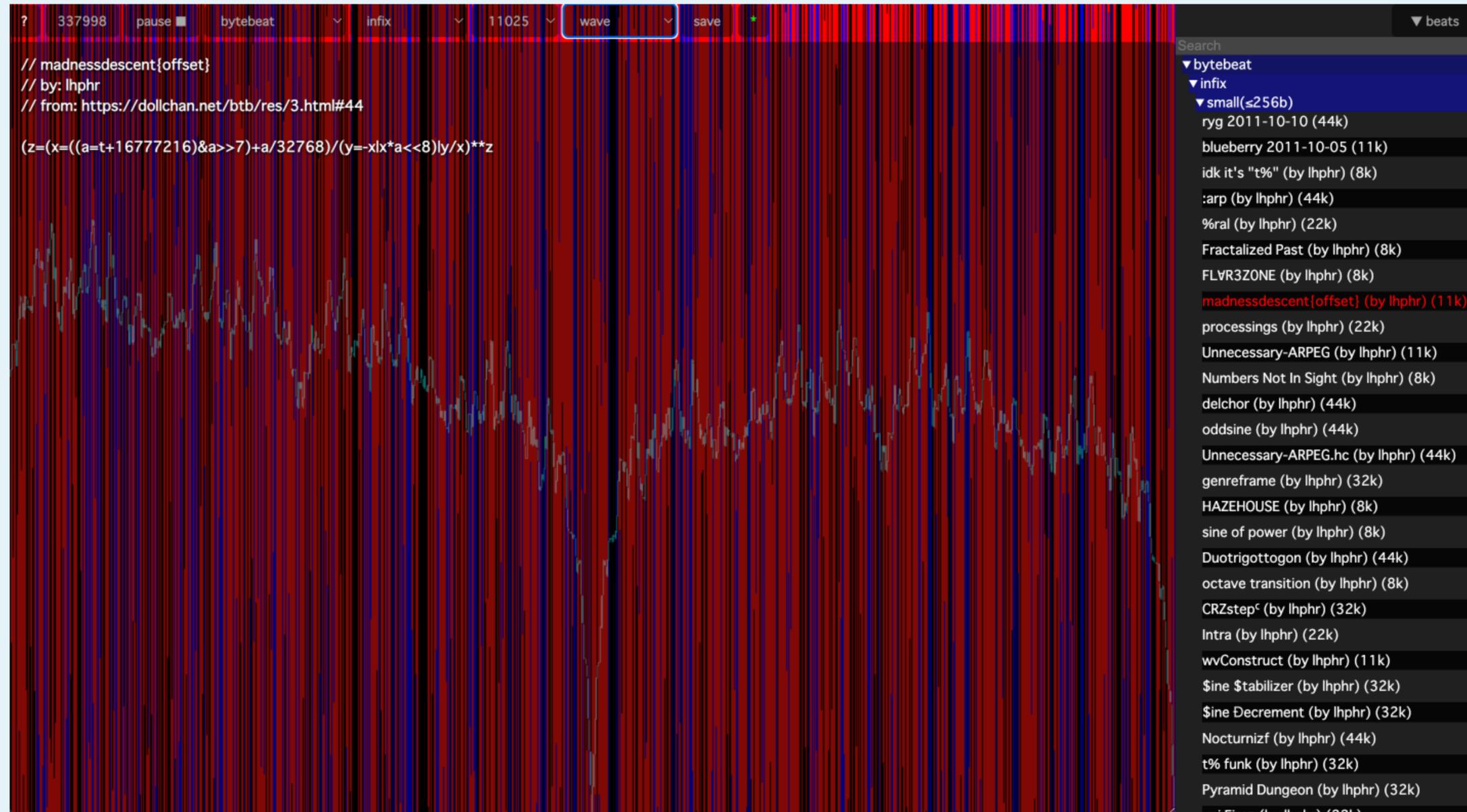
▼ October (2)

[Some deep analysis of one-line music programs.](#)

Algorithmic symphonies from

<http://countercomplex.blogspot.com/2011/10/some-deep-analysis-of-one-line-music.html>

# HTML5 Bytebeat



The screenshot displays the HTML5 Bytebeat player interface. The main area shows a waveform visualization with a red background and a blue line representing the audio signal. The waveform is complex and oscillatory. The interface includes a top bar with controls like 'pause', 'wave', and 'save'. Below the waveform, there is a code editor showing the following code:

```
// madnessdescent{offset}  
// by: lhphr  
// from: https://dollchan.net/btb/res/3.html#44  
  
(z=(x=((a=t+16777216)&a>>7)+a/32768)/(y=-xix*a<<8)ly/x)**z
```

On the right side, there is a search bar and a list of tracks. The tracks listed are:

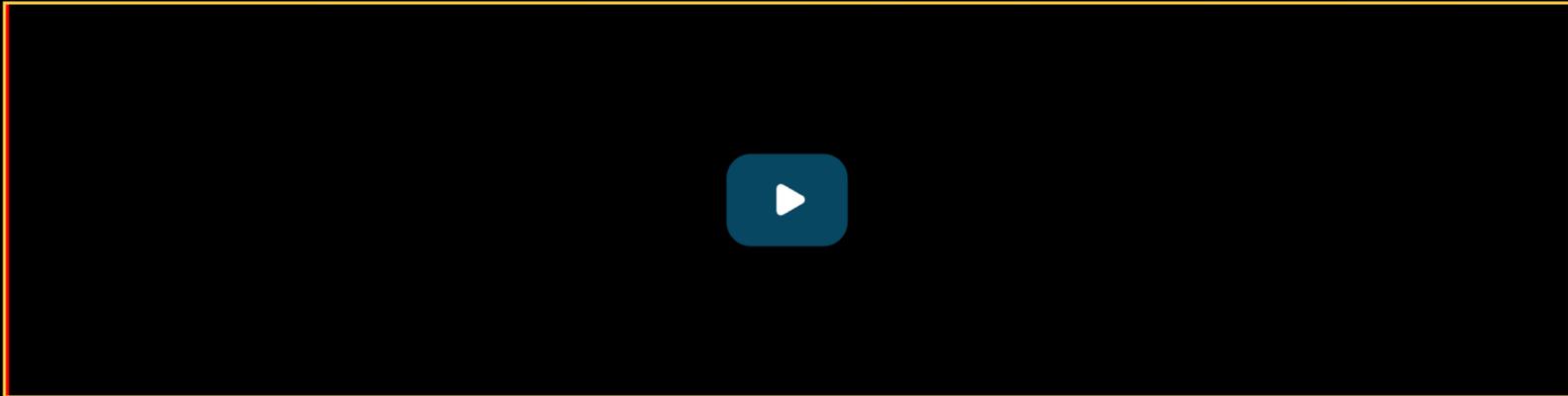
- bytebeat
- infix
  - small(≤256b)
    - ryg 2011-10-10 (44k)
    - blueberry 2011-10-05 (11k)
    - idk it's "t%" (by lhphr) (8k)
    - :arp (by lhphr) (44k)
    - %ral (by lhphr) (22k)
    - Fractalized Past (by lhphr) (8k)
    - FLVR3ZONE (by lhphr) (8k)
    - madnessdescent{offset} (by lhphr) (11k)
    - processings (by lhphr) (22k)
    - Unnecessary-ARPEG (by lhphr) (11k)
    - Numbers Not In Sight (by lhphr) (8k)
    - delchor (by lhphr) (44k)
    - oddsine (by lhphr) (44k)
    - Unnecessary-ARPEG.hc (by lhphr) (44k)
    - genreframe (by lhphr) (32k)
    - HAZEHOUSE (by lhphr) (8k)
    - sine of power (by lhphr) (8k)
    - Duotrigottogon (by lhphr) (44k)
    - octave transition (by lhphr) (8k)
    - CRZstep<sup>c</sup> (by lhphr) (32k)
    - Intra (by lhphr) (22k)
    - wvConstruct (by lhphr) (11k)
    - \$ine \$tabilizer (by lhphr) (32k)
    - \$ine Decrement (by lhphr) (32k)
    - Nocturnizf (by lhphr) (44k)
    - t% funk (by lhphr) (32k)
    - Pyramid Dungeon (by lhphr) (32k)
    - ui Firaz (by lhphr) (22k)

<https://greggman.com/downloads/examples/html5bytebeat/html5bytebeat.html>

# Bytebeat Composer

```
(t*(1+(5&t>>10))*(3+(t>>17&1?(2^2&t>>14)/3:3&(t>>13)+1))>>(3&t>>9))&[(t&4096?(t*(t^t%9)|t>>3)>>1:255)]
```

t 0 Waveform ▾ Bytebeat ▾ 8000 Hz / 1



▼ Info — about bytebeat

Bytebeat music (or one-liner music) was invented in September 2011. Simple bytebeats are often a piece of rhythmic and somewhat melodic music with no score, no instruments, and no real oscillators. It's simply an expression that defines a waveform as a function of time, processed (usually) 8000 times per second, resulting in an audible waveform with a 256-step resolution from silence (0) to full amplitude (256). If you put that formula into a program with a loop that increments time variable (t), you can generate the headerless unsigned 8 bit mono 8kHz audio stream on output, like in this application. Since these directly output a waveform, they have great performance in compiled languages and can often be ran on even the weakest embedded devices.

[History of bytebeat](#)

Original blog posts and videos from Viznut:

- [Blog posts #1](#)
- [Blog posts #2](#)
- [YouTube video #1](#)

<https://sarpnt.github.io/bytebeat-composer>

Node.js と FFmpeg で Bytebeat

# Node.jsでbytebeat

- Javascriptでバイナリデータを直接扱うのは意外と難しい
- Uint8Arrayという1バイトずつのデータを直接扱える配列の機能を使うことで実現できる
- 直接stdoutに書き込むのが少し大変なので、一度ファイルを経由してみよう

# bytebeat\_file.js

```
const fs = require("fs");
const sample_rate = 8000;
const seconds = 5;
const length = sample_rate * seconds;
const bytebeat = t =>
  (t >> 10 ^ t >> 11) % 5 * ((t >> 14 & 3 ^ t >> 15 & 1) + 1) * t % 99 + ((3 + (t >>
14 & 3) - (t >> 16 & 1)) / 3 * t % 99 & 64);
let t = 0;
//データを保存
const data = Uint8Array.from(
  { length: length },
  (v, _t) => {
    t += 1;
    return bytebeat(t);
  });
fs.writeFile("jsbytebeat.hex", data, _err => { });
```

これをnode bytebeat\_file.jsとして実行すると、jsbytebeat.hexというファイルができる

# bytebeat\_file.js

```
cat jsbytebeat.hex | ffplay -f u8 -i pipe:0 -ar 8k -ac 1
```

これで聞いてみよう

```
cat jsbytebeat.hex | ffmpeg -f u8 -ar 8k -i pipe:0 -c:a  
pcm_u8 -ac 1 -y bytebeat.wav
```

ffmpegを使えば改めてオーディオファイルに保存もできる

(-yは上書き保存を許すためのオプション)

# 実際どういう波形が出てるの？

- 波形の形はffmpegでwavに保存してaudacityで開けば見られる
- しかし、そもそもバイナリデータなのでテキストに簡易的にグラフを作ること  
もできる

# テキストファイルで擬似プロット

```
//bytebeat_file.jsの続き...  
//テキストファイルで擬似プロット  
let file = fs.createWriteStream('graph.txt');  
const bargraph = data.map(byte => {  
  let txt = "";  
  for (i = 0; i < byte; i++) {  
    txt += "X";  
  }  
  txt += "\n";  
  file.write(txt);  
});  
file.end();  
fs.writeFile(bargraph, "graph.txt", err => {});
```

出来上がったgraph.txtをエディタで開いてみよう



# bytebeat\_stream.js

```
const fs = require("fs");
const { setTimeout } = require('timers/promises');
const sample_rate = 8000;
const seconds = 5;
const length = sample_rate * seconds;
const bytebeat = t =>
  (t >> 10 ^ t >> 11) % 5 * ((t >> 14 & 3 ^ t >> 15 & 1) + 1) * t % 99 + ((3 +
  (t >> 14 & 3) - (t >> 16 & 1)) / 3 * t % 99 & 64);
let t = 0;
(async () => {
  while (true) {
    const data = Uint8Array.from({ length: length }, (v, _t) => {
      t += 1;
      return bytebeat(t);
    });
    process.stdout.write(data);
    await setTimeout(seconds / 1000.0);
  }
})()
```

node bytebeat\_stream.js| ffplay -f u8 -i pipe:0 -ar 8k -ac 1  
で無限に再生できるようになる

# 何が起きてるのか？

- 単純な式を少しずつ複雑にして行ってみよう

- $t$

- $t\%20$

- $t\&100$

- $t\%20 + t \%50$

足し算は波形同士の足し算として機能する！

- $t \& (t \gg 4)$

- $t \& (t \gg 4) + t\%5$